The KBSA Requirements/Specification Facet: ARIES

W. Lewis Johnson, Martin S. Feather

USC / Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695

David R. Harris

Lockheed Sanders
MER24-1583, P.O. Box 2034
Nashua, NH 03061-2034

Abstract

This paper describes a transformation-based software environment which supports the acquisition and validation of software requirements specifications. These requirements may be stated informally at first, and then gradually formalized and elaborated. The environment assists in the validation of formalized requirements by translating them into natural language and graphical diagrams, and testing them against a running simulation of the system to be built. Requirements defined in terms of domain concepts are transformed into constraints on system components. The advantages of this approach are that specifications can be traced back to requirements and domain concepts, which in turn have been precisely defined.

1 Introduction

We are building a requirements/specification environment called ARIES[1] which requirements analysts may use in evaluating system requirements and codifying them in formal specifications. We are creating this environment to help us address several roadblocks in providing knowledge-based automated assistance to the process of developing formal specifications. One of the principal roadblocks is that formal specification languages are difficult to use in requirements acquisition, particularly by people who are not experts in logic. ARIES provides tools for the gradual evolution of acquired requirements, expressed in hypertext and graphical diagrams, into formal specifications. The analysts invoke transformations to carry out this evolution; in general, support for rapid and coordinated evolution of requirements is a major concern. ARIES is particularly concerned with problems that arise in the development of specifications of large systems. Specification reuse is a major concern, so that large specifications do not have to be written from scratch. Mechanisms are provided for dealing with conflicts in requirements, especially those arising when groups of analysts work together. Validation techniques, including simulation, deduction, and abstraction, are provided, to cope with the problem that large specifications are difficult to understand and reason about.

ARIES is an intensively knowledge-based system. It incorporates knowledge about application domains, system components, and design processes, and supports analysts in applying this knowledge to the requirements analysis process. ARIES is not just a research prototype, but a plausible model for how to build knowledge-based tools that can handle large-scale software engineering problems.

The next section provides background and motivation for the ARIES effort. Section 3 describes the case studies of knowledge-based requirements acquisition we conducted as part of the project, and the lessons learned that motivated the functionality of the system. Section 4 then gives an overview of the mechanisms developed, particularly those supporting specification evolution.

2 Background

ARIES is a product of the ongoing Knowledge-Based Software Assistant (KBSA) program. KBSA, as proposed in the 1983 report by the US Air Force's Rome Laboratories [12], was conceived as an integrated knowledge-based system to support all aspects of the software life cycle. Such an assistant would support specification-based software development: programs would not be written in conventional programming languages, but instead would be written in an executable specification language, from which efficient implementations would be mechanically derived. In a

[1]ARIES stands for Acquisition of Requirements and Incremental Evolution of Specifications.

0-8186-2605-4/91 $1.00 © 1991 IEEE

complete KBSA system, and to some extent in ARIES as well, requirements analysis tools and implementation tools are integrated into a single environment, allowing analysts to perform exploratory prototyping during requirements analysis. Nevertheless, the design of ARIES does not preclude its use in situations where a full KBSA system is not available.

The ARIES effort builds on the results of earlier efforts at USC / ISI and Lockheed Sanders. Requirements analysis was addressed in Lockheed Sanders's Knowledge-Based Requirements Assistant [14]. ISI developed the Knowledge-Based Specification Assistant [27, 18, 17] to support specification construction, validation, and evolution.

The Requirements Assistant provided facilities for acquisition of informal requirements, entered as structured text and diagrams. It had a limited case-frame-based ability to assist in the formalization of informal text, by recognizing words in a lexicon of domain concepts. It allowed users to describe systems from different points of view, e.g., an "intelligent notepad" for informal text, data flow, state transition, and functional decomposition. The Requirements Assistant maintained an internal representation of the system being built which integrates these different views, as do certain other CASE tools such as STATEMATE [13]. It was able to generate DoD-STD-2167A-style requirements documents from its system descriptions.

The principal contribution of the Knowledge-Based Specification Assistant was the development of evolution transformations for specification modification [20]. The Specification Assistant also provided validation tools in the form of a paraphraser which translates specifications into English [33, 25], a symbolic evaluator for simulating the specification and proving theorems about it [6], and static analysis tools which automatically maintain and update analysis information as the specification is transformed [23].

2.1 The Requirements/Specification Work Product

ARIES assumes a model of software development in which there are multiple goals for requirements analysis. First, it should produce a software requirements specification (SRS), describing the characteristics of the system to be built. Davis, in his book *Software Requirements Analysis and Specification* [8], identifies a number of properties of a good SRS;it should be correct, unambiguous, complete, verifiable, consistent, understandable, modifiable, traceable, and annotated. We are in agreement with his list. However, textual documents are themselves but a means to achieve a more fundamental goal, namely communication of requirements to designers and stakeholders (end-users, procurement agents, etc.). Other communication media, such as diagrams, are useful to accomplish successful communication. Executable prototypes are another useful product, both to help communicate requirements and to validate the accuracy of those requirements. Finally, we assume that system requirements are not developed from scratch and thrown away. Instead, a goal of the requirements analysis process should be to develop generic requirements descriptions applicable to many possible systems in the application domain, and reuse such descriptions where they exist.

3 The Requirements/Specification Process

Over the past two years, we have grounded our work using two application domains — road traffic control and air traffic control. We are currently using ARIES to formalize the requirements for significant sections of the Federal Aviation Administration's Advanced Automation System (AAS) system specification [1]. So far, around 1500 concepts and requirements definitions have been entered into the system, of which 350 are specific to AAS, 160 are specific to an example road traffic control specification, and 1000 are reusable in nature. The experiments in specification development in these domains have led us to a number of conclusions regarding the nature of the specification development process.

3.1 Central issues

We believe that coordinating multiple users and viewpoints, capturing requirements that can be shared across systems, and sharing core concepts and knowledge across domains are central to supporting the requirements/specification process.

3.1.1 Coordinating multiple analysts and viewpoints

The requirements for future air traffic control systems are extremely detailed: system descriptions for the Advanced Automation System (AAS) run into the hundreds of pages. The work of specification must be divided among multiple analysts in order to be feasible. In our current analysis and formalization of sections of the AAS requirements, we find that the the FAA has identified particular functional areas for that system. These areas, including track processing, flight plan processing, and traffic management, seem to be good candidates for assignment to different analysts or analyst teams.

However, one important conclusion we have drawn is that a proper balance must be struck between coordinated and independent work of analysts. Requirements are not like program modules, that can be handed off to independent coders to implement. There is inevitably significant overlap between them. They may share a significant amount of common terminology between them. Requirements expressed in one functional area may have impact on other functional areas. In the AAS specification, we specified track processing, flight plan processing, and assignment of control separately. By comparing notes, we then found that flight plan information had an impact on how tracks are disambiguated, and that the process of handing off control of aircraft from one facility to the next had an impact on when flight plan information is communicated between facility computer systems. Figure 1 illustrates that these overlapping concerns become very apparent when viewed from the state transition perspective. Our approach to this issue has been to work on machine-mediated ways to support separation and subsequent merging of work products, rather than to force analysts to constantly coordinate whenever an area of potential common concern is identified.

Inconsistency is pervasive. Separate development of different requirements areas inevitably leads to inconsistencies. These inconsistencies are a natural consequence of allowing analysts to focus on different concerns individually. Although consistency is an important goal for the requirements process to achieve, we have concluded that it cannot be guaranteed and maintained throughout the requirements analysis process without forcing analysts to constantly compare their requirements descriptions against each other. Therefore, consistency must be achieved gradually, at an appropriate point in the specification development process. Nevertheless, it may not be possible to recognize all inconsistencies within a system description.

Multiple models must be supported. One place where inconsistencies need to be resolved is where multiple models are used. For example, when analysts specify radar processing requirements it is important to model the dynamics of aircraft motion to make sure that the system is able to track aircraft under normal maneuver conditions. When specifying flight plan monitoring, however, it is sufficient to assume that aircraft will move in straight lines from point to point, and change direction instantaneously, since the time required for a maneuver is very short compared to the time typically spent following straight flight paths. One way of resolving such conflicts is to develop a specialization hierarchy that relates these models to common abstractions.

3.1.2 Sharing requirements across systems

The designer of an air traffic control system must make sure that computers and human agents can together achieve the goals of air traffic control, i.e., to ensure the safe, orderly, and expeditious flow of air traffic. How this will be done by the AAS is to some extent determined by current air traffic control practice. Thus the next generation of controller consoles are being designed to simulate on computer displays the racks of paper flight strips that controllers currently use to keep track of flights. Yet although air traffic control practice is codified in federal regulations and letters of agreement, and is thus resistant to change, the division of labor between computer and human controller is expected to change over time. The FAA anticipates that new computer systems will gradually be introduced into the new air traffic control framework over the next twenty years, taking increasing responsibility for activities that are now performed by controllers.

Thus requirements descriptions for the AAS system must be shared across a series of future systems. If separate requirements documents are developed separately for each system, there is no assurance that the same requirements will be satisfied by each successive system. Furthermore, it is not sufficient simply to copy requirements from one requirements document to the next. Since some aspects of the system environment change and some aspects do not, it is difficult to take requirements expressed assuming one environmental context and transfer them to another environmental context. It therefore appears desirable to be able to represent overall requirements on air traffic control, without being forced to commit to particular computer systems satisfying those requirements. We can only conjecture whether or not this need to share requirements applies to other domains, but we believe that it will in any situation where successive versions of computer systems are anticipated, or where system maintenance may require extensive revision over time. We therefore believe it appropriate to focus ARIES on this class of domains and systems.

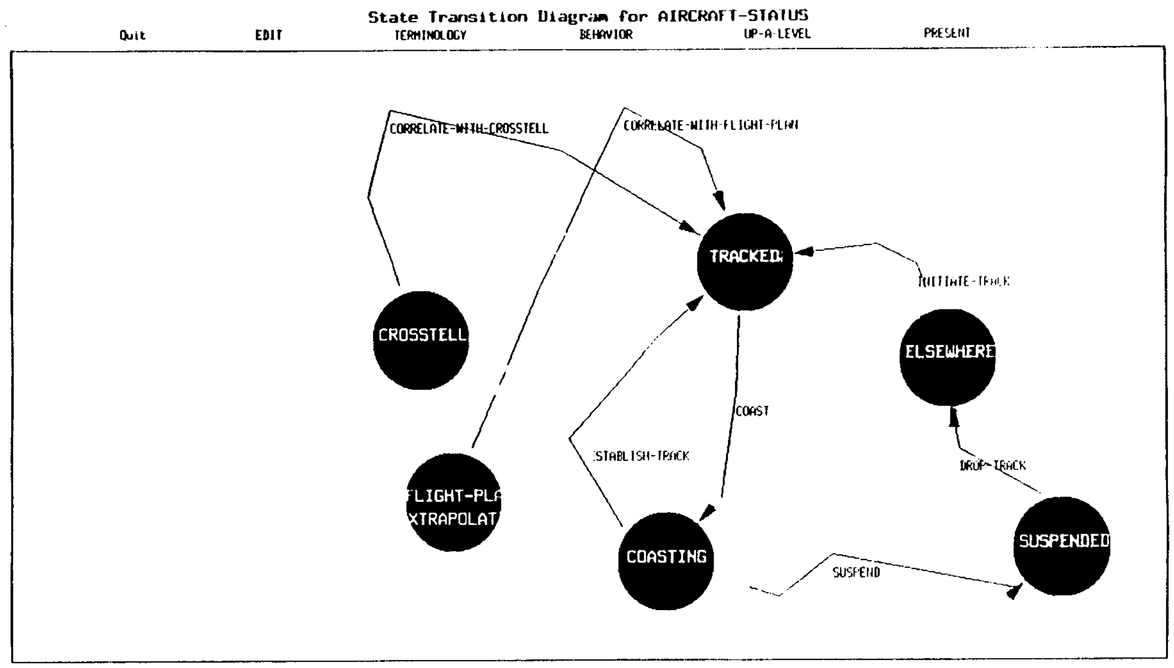

Figure 1: Specifications for tracking, flight plan processing, and handoff (crosstell in the diagram) are highly interrelated

3.1.3 Sharing across domains

Just as there are opportunities for sharing across systems in the same domain, there are opportunities for sharing across domains. The road traffic control problem shares certain characteristics with air traffic control: both problems are concerned with the maintenance of safe, orderly, and expeditious flow of vehicular traffic. They both assume a common body of underlying concepts, such as vehicles, sensors, spatial geometry, etc. We have been endeavoring to model such concepts so that the commonalities and differences across the two domains are captured.

3.2 Typical modifications

We are discovering important categories of specification modification. These categories are enabling us to provide adequate automation for all aspects of specification development and are providing insight into finding and encoding cornerstone decisions of a typical development.

- *Categorization by focus of change*

  Our experience has revealed that the choice of which evolution transformation to apply is often determined by the nature of the change we wish to achieve. We are identifying the important characteristics of specifications that people reason about and manipulate during design — modularity, entity-relationship, data flow, and control flow.

- *Informal to formal mappings*

  Analysts will frequently refer to existing documents for clarification or endorsement of requirements statements. The link between paragraphs in documents and resulting specifications either goes unstated or has only rudimentary machine mediation. Hyperstrings are used in ARIES to capture informal and semi-formal information.

- *General to specific mappings*

  One common technique is to define a concept first by placing it at a high point in the specialization hierarchies, and describing informally what additional properties the concept should have. After browsing the knowledge base further, specializations that are closer to the analyst's intent may be found, or else existing specializations may be adapted using evolution transformations. This allows the analyst to position the concept further down in the hierarchy.

- *Default removal*

  Another common incremental formalization is the elimination of defaults. A system description that is more detailed, so that defaulting is no longer necessary, can be viewed as more formal than one in which defaults are liberally employed.

- *Ideal to compromised*

In many cases the initial requirements are overly ideal, and in the course of development they must be carefully compromised to make them mutually compatible. Traceability must record when (and why) such compromises occur. Since compromises to requirements are evolutions, we use our same mechanism — evolution transformations — to carry them out, and record the applications of these transformations to provide traceability.

4 Mechanisms for Supporting Specification Evolution

In this section, we describe the major technical challenges we have undertaken to create ARIES. These challenges are explored in more detail elsewhere [21, 22, 20].

4.1 Folders

Requirements may be defined by specializing and adapting existing requirements in ARIES's knowledge base of common requirements; this makes it easier to define requirements quickly and accurately. *Folders* are used in ARIES to capture, separate, and relate bodies of requirements information. The analysts can control the extent to which folders share information, and gradually increase the sharing as inconsistencies are reconciled. ARIES places a heavy emphasis on codification and use of domain knowledge in requirements analysis. (See the accompanying paper, [15], for more detailed information on this important component of ARIES.)

4.2 Acquisition and review

The acquisition tools in ARIES aim to capture initial statements of requirements as simply and directly as possible. If requirements cannot be initially stated in a manner that is intuitive for the analyst or end-user, it is difficult to ensure that the requirements are correct. These actions are done through an interface called the Presentation Facility, which makes it possible to enter and view information through a variety of different notations. All notations that ARIES supports are views of the same underlying system description representation. The notations, which we call "presentations," include diagrams (such as data flow, state transition, functional decomposition, type taxonomy, relation taxonomy, event taxonomy), spreadsheets, formal specification text, and informal paraphrasing.

4.2.1 Supporting multiple presentations

The key technical challenge in supporting multiple presentations has been to develop a common internal representation, the ARIES Metamodel, that will easily map to the notations of stereotypical views of systems (e.g., data flow arcs, system functional decomposition, state transitions, predicate calculus-like formalisms). Some metamodel concepts are relatively separable and easy to handle. For example, the type, relation, and event taxonomic diagrams are generated from the internal representation in an obvious way. Other concepts are highly interrelated. States are relations which are derived from a designated relation which has a parameter varying over a finite set of values. State transitions are demon events that update the current-state — that is, change the value of the parameter and place some aspect of the system in a new state. In figure 1, initiate-track, correlate-with-crosstell, and the other arrows are all state-transition events. tracked and the other circles are relations for each of the possible states that an aircraft track can be in.

This means that arbitrary relations and events will not be presented in state transition diagrams, since they are not of the desired form. Instead, the state transition presentation only presents relations of class state-relation, i.e., those which are in a form similar to a current-state relation that would have been acquired as a state transition diagram. Conversely, state-transitions and state-relations will show up in other presentations. For example, a state-transition will appear in any RG presentation as a demon, and along with other demons will populate various event presentations.[2]

4.2.2 The presentation architecture

The ARIES Presentation Facility is an architecture for defining interactive presentations linked to the ARIES Metamodel. It is implemented in CLX and CLUE, on top of X windows, and is operational on both the TI Explorer and the Sun. Each presentation definition includes a declarative description of the metamodel relations which are used to establish and link presentation pieces, and the editing and navigation actions (associated either with a presentation piece or the entire presentation). Editing actions match effect descriptions of transformations. Once the analyst edits

[2] RG, or Reusable Gist [19], is our principal formal specification language, based on the earlier Gist language but incorporating constructs to support reuse, and with a syntax based in part on other current high-level programming languages, such as Refine[28].

a presentation, ARIES searches for and applies the evolution transformations which can make the required change. The ARIES presentation framework makes it possible to construct powerful presentations combining text and graphics generation capabilities.

4.3 Analysis and simulation

Analysis tools include a constraint propagation engine and an incremental static analyzer. Analysis tools are important in order to check for completeness and consistency. A constraint mechanism, derived from Steele's Constraint Language [32], has been incorporated into ARIES for general maintenance of constraints — bidirectional propagation, contradiction detection, retraction, and explanation. This mechanism is essential where there are interacting design properties (e.g., interplay between performance characteristics) and developers can use assistance in identifying when an interaction of requirements may not be achievable. An incremental static analyzer, a version of the static analyzer developed for the Specification Assistant [23], maintains calling and type information for the system description as it is being edited. It also does such things as detect specification freedoms which must be removed temporarily before simulation can be performed.

Simulation tools are useful in order to observe the behavior of a proposed system or its environment, in order to determine appropriate parameters for requirements or to discover unexpected or erroneous behavior. Simulations are constructed by means of a specially modified compiler which translates a subset of the ARIES Metamodel into Lisp and AP5, an in-core relational database [7]. Events described in the specification can compile either into ordinary Lisp functions, or into task objects to be scheduled by the simulator's task scheduler. Functional requirements in the form of invariants are compiled into rules which notify the analyst if and when they are violated [3].

Successful simulation analysis depends crucially upon the model of the system and environment chosen for simulation. When attempting to answer a specific validation question, it is useful to remove from consideration those features of the system which are not relevant to the question. Otherwise the simulation will generate volumes of useless information.

Kevin Benner in our group is currently investigating which abstractions are most suitable for which kinds of analysis tasks. He is developing evolution transformations which construct the abstractions and designing the simulator to execute these abstractions. Together these form a powerful set of capabilities for specification validation.

4.4 Evolution transformations

As part of our earlier Specification Assistant work, we built a sizable library of evolution transformations, that is, transformations whose very purpose is to *change* the meaning of the specification to which they are applied. Like conventional correctness-preserving transformations, they blend computer power — the ability to conduct repeated, mechanical operations rapidly and reliably — and human intuition — knowing which transformation to apply when.

A major goal of the ARIES project has been to support the user in selecting evolution transformations from a library, and in applying them. This library constitutes reusable knowledge about the *process* of requirements analysis, which complements the knowledge about the inputs and outputs of this process, i.e., knowledge of domains and systems.

The representation of specification concepts enables efficient and effective modification of the semantic content of complex specifications. Having identified specification characteristics, we then chose a common representation for them, semantic nets — nodes connected by links, where the types of the nodes and links determine which characteristic they represent. For example, in the entity-relationship model, procedures and types are represented by nodes; the type of a procedure's formal parameter is represented by linking the node representing that procedure with the node representing that type. Changes to the specification induce the corresponding changes on these semantic net representations of the specification's characteristics. Each change can be expressed as a combination of creating and destroying nodes, and inserting and removing links between nodes. We have identified frequently recurring composites of these operations, for example, *splice* removes a direct link between two nodes, A and B say, and replaces it with two links via an intermediary, C say, so that A is linked to C and C is linked to B.

We have characterized each evolution transformation in terms of the effects it induces on the semantic net representation of each of the above categories. Splice-Data-Accesses is an example of a transformation that performs a splice along the information flow dimension. Likewise, an evolution transformation that introduces an intermediate specialization of some concept (e.g., given a specification containing type person and type airline-pilot, a specialization of person, we might introduce an intermediate type employed-person) is characterized as inducing a *splice* upon the specialization link structure. Similarly, an evolution transformation that wraps a statement inside a conditional is also characterized as inducing a *splice*, but upon the control-flow structure (the control flow link that led into the original statement now leads into the surrounding conditional statement, and there is a link from the conditional to the original statement).

4.5 Reconciling conflicting views

ARIES permits views of specifications to diverge, via different folders. Although divergence is important at certain stages of the analysis process, such divergences must eventually be reconciled.

One technique that we have explored to facilitate reconciliation is the process of merging parallel elaborations [9]. Feather analyzed a restricted case of reconciliation, where different views of the specification are all derived by transformation from a common root specification, which describes the system in a very abstract way. This technique attempts to combine the various transformations into a linear sequence. By analyzing the transformations, their applicability conditions, and what they apply to, it is possible in many cases to determine automatically whether transformations applied to different views may interfere with each other.

The approach that we envision for ARIES centers on gradual elimination of differences between the conceptual models, regardless of their origin. If two members of an analysis team are using conflicting definitions of the same concept, they will each employ transformations step by step to eliminate those differences. In some cases this will involve having each analyst distribute the transformations that they employed so that the other analyst can employ them as well. As differences are resolved, specification components can be gradually promoted to the project-shared folders. In those cases where an analyst has employed a model that is more detailed than necessary for the shared model, abstraction transformations may be employed to reduce the detail to the level shared throughout the project.

5 Related Work

The evolutionary approach to requirements specification has a number of precursors. Burstall and Goguen argued that complex specifications should be put together from simple ones, and developed their language CLEAR to provide a mathematical foundation for this construction process [5].

Goldman observed that natural language descriptions of complex tasks often incorporate an evolutionary vein — the final description can be viewed as an elaboration of some simpler description, itself the elaboration of a yet simpler description, etc., back to some description deemed sufficiently simple to be comprehended from a non-evolutionary description [11].

Fickas suggested the application of an AI problem-solving approach to specification construction [10]. Fundamental to his approach is the notion that the steps of the construction process can be viewed as the primitive operations of a more general problem-solving process, and are hence ultimately mechanizable. Continuing work in this direction is reported in [31] and [2].

In the Programmer's Apprentice (PA) project (see [30]), the aim is to build a tool which will act as an intelligent assistant to a skilled programmer. In their approach, programs are constructed by combining algorithmic fragments stored in a library. These algorithmic fragments are expressed using a sophisticated plan representation, with the resulting benefit of being readily identifiable and combinable. Their more recent work on supporting requirements acquisition (the "Requirements Apprentice," [29]) addresses the early stages of the software development process, operating on representations of requirements. The PA and ARIES approaches are closely related. Many evolution transformations instantiate cliches as part of their function. We are currently exploring ways of making these cliches more explicit in our transformation system.

Karen Huff has developed a software process modeling and planning system that is in some ways similar to ours [16]. Her GRAPPLE language for defining planning operators influenced our representation of evolution transformations. Conversely, her meta-operators applying to process plans were influenced by our work on evolution transformations.

Kelly and Nonnenmann's WATSON system [24] constructs formal specifications of telephone system behavior from informal scenarios expressed in natural language. Their system formalizes the scenarios and then attempts to incrementally generalize the scenario in order to produce a finite-state machine. Acquisition from scenarios is a useful complement to the work we are doing and Benner in our group is currently investigating this area further [4].

The PRISMA project [26] is also a system for assisting in the construction of specifications from requirements. There is striking similarity between their approach and ours — the use of multiple presentations, and an underlying semantic-net formalism. They have clearly thought about and developed heuristics to operate on or between presentations, an aspect that we have only recently begun to address. Conversely, we have provided much more support for evolution.

6 Summary

ARIES provides a variety of capabilities to support the process of requirements acquisition and analysis. These capabilities include acquisition, review, evolution support, analysis, and reuse support. These are intended to help analysts satisfy the conflicting goals of software requirements specification in a gradual and systematic way. ARIES as a whole focuses on the problems of describing systems from different viewpoints, and reconciling different viewpoints.

By building the ARIES prototype, we have been able to identify and offer solutions for many of the significant challenges which must be met in making knowledge-based requirements and specification development environments a reality. Specifically we have concentrated on supporting reuse of large domain independent and dependent knowledge-bases, providing multi-presentation acquisition along with significant automation support in the form of evolution transformations, specification analysis, simulations. We have developed mechanisms around many requirements support features including folders, reuse techniques, acquistion and review, analysis and simulation, evolution transformations, and traceability. Work on the project is ongoing; most of the capabilities envisioned for the system are already in place, but much work remains to be done.

7 Acknowledgements

We wish to thank Kevin Benner for comments on this paper. Charles Rich has provided important guidance to this work. We would like to acknowledge current and previous members of the ARIES project: Jay Myers, K. Narayanaswamy, Jay Runkel, and Lorna Zorman. This work was sponsored in part by the Air force Systems Command, Rome Air Development Center, under contracts F30602-85-C-0221 and F30602-89-C-0103. It was also sponsored in part by the Defense Advanced Research Projects Agency under contract no. NCC-2-520. Views and conclusions contained in this paper are the authors' and should not be interpreted as representing the official opinion or policy of the U.S. Government or any agency thereof.

References

[1] Federal Aviation Administration. *Advanced Automation System: System Level Specification, FAA-ER-130-005G*, April 1987.

[2] J.S. Anderson and S. Fickas. A proposed perspective shift: Viewing specification design as a planning problem. In *Proceedings of the 5th International Workshop on Software Specification and Design, Pittsburgh, Pennsylvania*, pages 177–184. Computer Society Press of the IEEE, May 1989.

[3] K. Benner. Using simulation techniques to analyze specifications. In *Proceedings of the 5th KBSA Conference*, Rome, NY, 1990. Data Analysis Center for Software.

[4] K. Benner and W.L. Johnson. The use of scenarios for the development and validation of specifications. In *Proceedings of the Computers in Aerospace VII Conference*, Monterey, CA, 1989.

[5] R.M. Burstall and J. Goguen. Putting theories together to make specifications. In *Proceedings of the Fifth International Conference on Artificial Intelligence*, pages 1045–1058, August 1977.

[6] D. Cohen. Symbolic execution of the gist specification language. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 17–20. IJCAI, 1983.

[7] D. Cohen. *AP5 Manual.* USC-Information Sciences Institute, June 1989. Draft.

[8] A.M. Davis. *Software Requirements Analysis and Specification.* Prentice Hall, Englewood Cliffs, N.J., 1990.

[9] Martin S. Feather. Constructing specifications by combining parallel elaborations. *IEEE Transactions on Software Engineering*, 15(2):198–208, February 1989. Available as research report # RS-88-216 from ISI, 4676 Admiralty Way, Marina del Rey, CA 90292.

[10] S. Fickas. A knowledge-based approach to specification acquisition and construction. Technical Report 86-1, CS Dept., University of Oregon, Eugene, 1986.

[11] N.M. Goldman. Three dimensions of design development. In *Proceedings, 3rd National Conference on Artificial Intelligence, Washington D.C.*, pages 130–133, August 1983.

[12] C. Green, D. Luckham, R. Balzer, T. Cheatham, and C. Rich. Report on a knowledge-based software assistant. In *Readings in Artificial Intelligence and Software Engineering.* Morgan Kaufmann, Los Altos, CA, 1986.

[13] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring. Statemate: A working environment for the development of complex reactive systems. In *Proceedings of the 10th Intl. Conf. on Software Engineering*, 1988.

[14] D. Harris and A. Czuchry. The knowledge-based requirements assistant. *IEEE Expert*, 3(4), 1988.

[15] D. Harris and W.L. Johnson. Reuse of requirements knowledge. In *Proceedings of KBSE 6*, 1991. submitted.

[16] K.E. Huff and V.R. Lesser. The GRAPPLE plan formalism. Technical Report 87-08, U. Mass. Department of Computer and Information Science, April 1987.

[17] W.L. Johnson. Deriving specifications from requirements. In *Proceedings of the 10th International Conference on Software Engineering*, pages 428–437, 1988.

[18] W.L. Johnson. Specification as formalizing and transforming domain knowledge. In *Proceedings of the AAAI Workshop on Automating Software Design*, 1988.

[19] W.L. Johnson and M.S. Feather. Reusable gist language description. Available from USC / ISI, 1991.

[20] W.L. Johnson and M.S. Feather. Using evolution transformations to construct specifications. In *Automating Software Design.* AAAI Press, 1991.

[21] W.L. Johnson, M.S. Feather, and D.R. Harris. Applying domain and design knowledge to requirements engineering. *IEEE Office Knowledge Newsletter*, 1991. in press.

[22] W.L. Johnson, M.S. Feather, and D.R. Harris. Integrating domain knowledge, requirements, and specifications. *Journal on System Integration*, 1991. in press.

[23] W.L. Johnson and K. Yue. An integrated specification development framework. Technical Report RS-88-215, USC / Information Sciences Institute, 1988.

[24] V.E. Kelly and U. Nonnenmann. Reducing the complexity of formal specification acquisition. In *Proceedings of the AAAI-88 Workshop on Automating Software Design*, pages 66–72, 1988.

[25] J.J. Myers and W.L. Johnson. Towards specification explanation: Issues and lessons. In *Proceedings of the 3d Knowledge-Based Software Assistant Conference*, 1988.

[26] C. Niskier, T. Maibaum, and D. Schwabe. A look through PRISMA: Towards pluralistic knowledge-based environments for software specification acquisition. In *Proceedings, 5th International Workshop on Software Specification and Design, Pittsburgh, Pennsylvania, May*, pages 128–136. Computer Society Press of the IEEE, 1989.

[27] The KBSA Project. Knowledge-based specification assistant: Final report. unpublished, 1988.

[28] Reasoning Systems, Palo Alto, CA. *Refine User's Guide*, 1986.

[29] H.B. Reubenstein and R.C. Waters. The requirements apprentice: Automated assistance for requirements acquisition. *IEEE Trans. on Software Engineering*, 17(3):226–240, March 1991.

[30] C. Rich. *The Programmer's Apprentice.* ACM Press, Baltimore, MD, 1990.

[31] W.N. Robinson. Integrating multiple specificationss using domain goals. In *Proceedings, 5th International Workshop on Software Specification and Design, Pittsburgh, Pennsylvania, May*, pages 219–226. Computer Society Press of the IEEE, 1989.

[32] G.L. Jr. Steele. The definition and implementation of a computer programming language. Technical Report 595, MIT Artificial Intelligence Laboratory, 1980.

[33] W. Swartout. Gist english generator. In *Proceedings of the National Conference on Artificial Intelligence.* AAAI, 1982.